(19) Europäisches Patentamt
Europ an Patent Office
Office européen des brevets

(11) Publication number: **0 288 606 B1**

(12) # EUROPEAN PATENT SPECIFICATION

(45) Date of publication of patent specification: **21.07.93**

(51) Int. Cl.5: **G06F 12/02**, G06F 12/08, G06F 12/10

(21) Application number: **87118538.5**

(22) Date of filing: **15.12.87**

(54) **Computer system employing a CPU having two mutually incompatible addressing modes.**

(30) Priority: **27.03.87 US 30789**

(43) Date of publication of application:
02.11.88 Bulletin 88/44

(45) Publication of the grant of the patent:
21.07.93 Bulletin 93/29

(84) Designated Contracting States:
AT BE CH DE ES FR GB GR IT LI LU NL SE

(56) References cited:
EP-A- 0 208 429
US-A- 4 128 875
US-A- 4 296 468

(73) Proprietor: **International Business Machines Corporation**
Old Orchard Road
Armonk, N.Y. 10504(US)

(72) Inventor: **Bealkowski, Richard**
1401 Hummingbird Drive
Delray Beach Florida 33444(US)
Inventor: **Dayan, Richard Alan**
830 73rd Street NE
Boca Raton Florida 33433(US)
Inventor: **Doria, David Joseph**
23396A SW 53rd Avenue
Boca Raton Florida 33433(US)
Inventor: **Kinnear, Scott Gerard**
9005 Saddlecreek Drive
Boca Raton Florida 33434(US)
Inventor: **Krantz, Jeffrey Issac**
9695A Boca Gardens Pky.
Boca Raton Florida 33434(US)
Inventor: **Liverman, Robert Britton**
6101 Friar's-Walk Place
Raleigh North Carolina 27609(US)
Inventor: **Sotomayor, Guy Gil**
2104 Carrol Circle
West Palm Beach Florida 33415(US)
Inventor: **Williams, Donald Dorsey**
940 NW 4th Avenue
Boca Raton Florida 33432(US)
Inventor: **Vaiskauckas, Gary Anthony**
6510 Timber Lane
Boca Raton Florida 33433(US)

(74) Representative: **Killgren, Neil Arthur**
IBM United Kingdom Limited Intellectual Property Department Hursley Park
Winchester Hampshire SO21 2JN (GB)

EP 0 288 606 B1

Description

The present invention relates to a computer system employing a processor having more than one addressing mode. More specifically, the invention pertains to such a computer system which is capable of running the same applications program in each of plural addressing modes.

Microcomputers (personal computers) are being required to perform more and more complex data processing tasks while not suffering a degradation in response speed. On the other hand, in order to perform the more sophisticated tasks, additional device drivers, networking programs, host attachment programs, session managing programs, etc., must be loaded into the available memory space. The amount of space left for the user's applications programs is thus shrinking, forcing undesirable trade-offs to be made among storage, performance, and function.

To remedy this problem, microprocessors have lately become available in which the amount of addressable memory has been greatly expanded. For example, for the Intel Corporation 8088/8086 microprocessor (hereinafter a microprocessor will be referred to for convenience as a "CPU" - central processing unit), the amount of addressable memory is about 1 MB (MegaBytes), while for the newer 80286 CPU, about 16 MB can be addressed.

The 80286, however, employs two different and mutually incompatible addressing modes. The first mode, termed the "real" mode, is exactly the same addressing mode employed in the 8088/8086 CPU, and hence programs written for a machine employing the 8088/8086 CPU, such as the vast array of software written for the IBM Personal Computer and compatibles, can be run in the real mode since the same BIOS (Basic Input-Output System) can be used directly. In the real mode, however, since the addressing mode is in fact the same as for the 8086/8088, the amount of addressable memory is still limited to about 1 MB.

The second mode, termed the "protected" mode, employs a different memory addressing scheme, and with this scheme can address up to about 16 MB of memory. However, because the addressing mode is indeed different, the earlier BIOS cannot be used successfully, and hence computers which have employed the 80286 CPU have not been able to simultaneously take advantage of the increased amount of available memory in the protected mode and run software written for the 8086/8088 CPU.

Fig. 1 shows a memory map of a typical microcomputer application employing an 80286 CPU and showing an example of how the memory may be organised. Memory addresses in the range of 0 KB (KiloBytes) to 40 KB are taken up by the BIOS (Basic Input-Output System) and OS (Operating System), the most famous examples being PC DOS and MS DOS marketed by Microsoft Corporation. The user is allocated the space from 40 KB to 640 KB in both the real and protected mode. The video buffers occupy 640 KB to 752 KB, and feature and planar ROMs (Read-Only Memories) 752 KB to 1 MB. This is all the memory that can be addressed in the real mode. In the protected mode though 15 MB of additional addressable memory space is available to the user.

To better understand the problem solved by the invention, the two addressing modes will now be described in more detail.

In both the 8088/8086 CPUs and in the real mode in the 80286 CPU, physical memory is addressed directly using 32-bit pointers. As shown in Fig. 2, each 32-bit pointer is composed of a 16-bit offset (bits 0 to 15) and a 16-bit segment (bits 16 to 31). The memory is divided into 64 KB segments, and each of the 16-bit segment values of the pointers corresponds directly to one of these 64 KB segments in memory. That is, pointer segment n, multiplied by 2^4 (equivalently, shifted one place in hexadecimal), directly indicates the address of the first eight-bit byte of data in segment n of the memory, namely, the boundary between segments n-1 and n in physical memory. The offset, on the other hand, indicates a displacement from the boundary between segments n-1 and n.

As indicated by the diagram of Fig. 3, to obtain the 20-bit value which directly addresses a given byte location (operand address) in physical memory, the segment value is multiplied by 2^4 and added to the offset value. This 20-bit address is applied directly to the memory as an address.

In the protected mode of the 80286, the BIOS does not use the physical memory in the form of segments and offsets. Moreover, the memory is not divided up into 64 KB segments. Instead, "virtual" memory addressing is employed in which the addresses do not correspond directly to distinct locations in physical memory. To allow for more efficient use of the available memory space while still retaining relative ease of addressing, the memory is again divided into segments, but the segments may be of variable lengths. Generation of the actual physical addresses is done internally to the 80286 CPU, out of reach of the user and BIOS.

The protected addressing mode will be explained in more detail with reference to Fig. 4 of the drawings.

As in the case of the 8088/8086 and real mode, the BIOS addresses memory using a 32-bit pointer. In the protected mode, the lower 16 bits (bits 0 to 15) of the pointer are also referred to as an offset. Because its function is different than that of the segment in the 8088/8086 and real mode the upper

16-bit portion of the pointer is termed a "selector". Instead of merely multiplying it by 2^4 and adding it to the offset, the selector is used as a pointer to a segment descriptor contained in a segment descriptor table, which is assembled in a predetermined area of the physical memory. Each segment descriptor contains a 24-bit value, which indicates the base address (lower boundary address) of the corresponding segment in physical memory. To obtain the actual address of a desired operand in physical memory, the 24-bit segment descriptor value retrieved by the selector is added to the offset.

Because the selectors employed in the protected mode thus have a function which is much different than and nonequivalent to that of the segments in the 8088/8086 and real mode, the BIOS designed for the 8088/8086 and real mode, which uses segments in addressing, cannot operate directly in the protected mode, making it impossible to run in the protected mode programs written to use the BIOS developed for the 8088/8086 and real mode. While such programs can of course be run on the 80286 CPU in the real mode, it is a disadvantage not to also be able to run them in the protected mode.

According to the invention there is provided a computer system comprising a central processing unit (CPU) and a memory, said CPU having first and second mutually incompatible modes for addressing said memory characterised by a basic input/output system (ABIOS, Fig. 5) adapted to set up first and second tables in said memory containing functionally equivalent pointers compatible respectively with said first and second modes to predetermined regions in the memory and devices attached to the system, said basic input output system being further adapted to access the first table under the first CPU mode and to access the second table under the second CPU mode.

There is further provided a method of operating a computer system having a memory and a CPU capable of operating in either of two mutually incompatible memory addressing modes characterised by the steps of: allocating space in said memory for first and second common data areas corresponding respectively to the first and second of said modes; loading said first command data area with a table containing first pointers to predetermined areas of the memory and to devices attached to the system, said first positions comprising first mode addresses; and loading said second command data area with a table containing pointers functionally equivalent to those of the first table, but comprising second mode addresses; whereby the CPU subsequently refers to said first table for first mode addressing and said second table for second mode addressing.

In the case of real and protected modes as discussed above, the pointers for the real mode common data area are each composed of a segment and an offset, and those of the protected mode common data area are composed of selectors and offsets. The offsets of the pointers in the protected mode table which point to addresses external to the common data area are identical in value to those of corresponding pointers in the real mode table, while the selectors point to segment descriptors identifying the physical addresses of the segments of the corresponding pointers.

An embodiment of the invention will now be described, by way of example, with reference to the accompanying drawings, in which:

FIG. 1 is a memory map of an example of a computer system employing a CPU having real and protected addressing modes;

FIG. 2 shows a portion of a main memory in the real mode and illustrates how the memory is addressed using offsets and pointers.

FIG. 3 depicts how segments and offset values are manipulated and added to produce physical memory addresses in the real mode;

FIG. 4 shows a portion of a main memory in the protected mode and illustrates how the memory is addressed using offsets and pointers;

FIG. 5 is a diagram showing how the BIOS code is arranged in accordance with the present invention;

FIG. 6 is a map of a bimodal common data area in accordance with the invention;

FIG. 7 is a flowchart showing in detail how the bimodal common data area illustrated in FIG. 6 is assembled;

FIG. 8 is a flowchart showing an example of how a device block pointer is obtained for a given logical ID; and

FIG. 9 is a flowchart illustrating how a request is made to the BIOS in the computer system of the invention.

Referring first to the diagram of FIG. 5, there is shown therein a memory map for the BIOS code employed in a computer system embodying the invention.

The BIOS code is composed of three parts: POST, CBIOS and ABIOS. POST (Power-On Self Test) is used for performing initial self testing and other basic start-up functions, including extracting the boot record from the system disk and subsequent loading of the operating system into memory. CBIOS (Compatibility Basic Input-Output System) contains the BIOS used by applications programs for performing input-output operations (transfers of data to and from memory, peripherals, etc.) in the real mode only and in a single-tasking environment. ABIOS (Advanced Basic Input-Output System) contains the BIOS used by applications

programs for performing input-output operations in a bimodal, multi-tasking environment.

In the present system, the ABIOS is given the capability of operating either in the real mode or the protected mode. The way in which this is done is for the ABIOS to assemble bimodal CDAs (Common Data Areas), one for the real mode and one for the protected mode. The entries contained in the two CDAs are identical in function, but in the real mode CDA the pointers are described in terms of segments and offsets, while in the protected mode CDA they are described in terms of selectors and offsets. If the operating system intends to execute BIOS only in the real mode, then only the real mode CDA need be assembled and used, whereby BIOS-controlled data transfers for applications programs take place in the previously known manner for operations in the real mode. Before the operating system can execute BIOS in the protected mode, however, the protected mode CDA must be assembled. By use of the protected mode CDA, since it is functionally identical to the real mode CDA, programs written for the real mode can successfully be run in the protected mode at the user's option. Hence, by providing the bimodal CDAs, the ABIOS code is essentially "transparent" to the mode in which the user has selected to run the CPU. The result is mode-independent addressing for the applications programs.

An example of bimodal CDAs is shown in accompanying FIG. 6. The following abbreviations are used in FIG. 6 for simplicity:

LogicalID (LID) - Each LID corresponds to and identifies a requested device. Each device available to ABIOS has a LID associated therewith.

Device Block (DB) - The DB is a working storage area allocated by the operating system which contains hardware port addresses, interrupt levels, and device state information.

Function Transfer Table (FTT) - The FTT is a permanent storage area allocated by the operating system and which contains the pointers to each ABIOS function routine.

ABIOS Data Pointer (Data Ptr) - The data pointers supply the ABIOS with addressability to particular portions of memory in the bimodal environment. Examples are the pointers to the video buffers.

As can readily be appreciated from FIG. 6, in general, the entries in the two CDAs are entirely identical in function and in their place within the respective tables; the only difference is that the pointers in the real mode CDA are composed of segment and offset values, and the pointers in the protected mode CDA are composed of selectors and offsets. Thus, by merely employing the CDA corresponding to the present operating mode of the CPU, so far as the remainder of the BIOS, the operating system, and the applications programs are concerned, all BIOS operations are performed in the identical manner between the two modes. That is, as illustrated in FIG. 6, the corresponding DB pointer LID n in the two tables both point to the same DB LID n, the corresponding data pointer 0 in the two tables point to the same identical location in memory, and the same function m pointers point to the same identical function m.

FIG. 7 is a flowchart describing in detail the manner in which the two CDAs are assembled.

From the START point, the ABIOS is called to build the systems parameters table in step 20. In step 21, ABIOS is called to build the initialisation table. Next, in step 22, memory space is allocated for the real mode CDA, including the DBs, FTTs, and data pointers. In subsequent step 23, an initial DB (InitDB) routine is called for initial table entries to build the FTT, DB, etc. Those having familiarity with the BIOS used in the IBM Personal Computer will understand the further details of how steps 20 to 23 are implemented.

After it has been determined in step 24 that all entries in the real mode CDA table are complete, in step 25, memory space is allocated for the protected mode CDA. In following step 26, an offset portion of a DB pointer in the real mode CDA is copied directly to the corresponding entry in the protected mode CDA. For the offset value copied in step 26, in steps 27 and 28, a selector is allocated which points to a segment descriptor identifying the physical base address of the segment of the respective pointer in the real mode CDA. In step 29, the selector is copied to the corresponding position in the protected mode CDA to complete the pointer. Thus, when the selector is processed in the CPU during operation in protect mode, the segment specified by the real mode segment address is accessed.

With reference to steps 30 to 34, an FTT pointer is assembled in a slightly different manner than the DB pointers in steps 26 to 29 since the FTT pointers point to other pointers within the CDA rather than addressed external to the CDA. In step 30, a protected mode FTT is allocated and, in step 31, a copy of the offset of the corresponding real mode FTT pointer is copied to the protected mode FTT pointer. In subsequent step 32, a determination is made as to whether a protected mode selector has been allocated. If not, in step 33, a selector is allocated which points to a segment descriptor identifying the base address of the segment containing the FTT. In step 34, the selector is copied to the selector portion in the protected mode CDA.

Similar to the assembly of the DB pointer in steps 26 to 29, in steps 35 to 38, a corresponding function pointer is assembled in the protected

mode CDA.

Step 39 tests to determine if all pointers of the FTT have been completed. Steps 35 to 39 are repeated until all pointers of the FTT have been assembled. Following step 40 tests to determine whether all logical IDs, DB pointers, and FTT pointers have been completed. If not, the process loops back to step 26, and steps 26 to 40 are repeated until all logical IDs, DB pointers, and FTT pointers have been completed.

To finish the protected mode CDA, in step 41, the data pointers (e.g., pointers to the video buffer) are assembled. This is done in the same fashion as above. In step 41, the offset portion of a data pointer from the real mode CDA is copied to the corresponding entry in the protected mode CDA, and in step 42, a selector is allocated pointing to a segment descriptor identifying the physical base address of the segment in the corresponding real mode CDA entry. The selector is copied into the protected mode CDA in step 43. In step 44, it is determined if all data pointers have been completed. If not, the process loops back to step 41, whereupon steps 41 to 44 are repeated until all data pointers have been finished. At that time, assembly of both CDAs is completed.

To show an example of how the assembled CDA is used by the ABIOS, FIG. 8 is a flowchart showing an example of how a DB pointer is obtained from the bimodal CDA for a given logical ID.

First, the "current" CDA anchor pointer and the corresponding logical ID are accessed in steps 60 and 61. In step 62, the logical ID is multiplied by 2^3 (=8) to obtain the offset for the DB pointer. Then, in step 63, the DB pointer can be accessed.

FIG. 9 is a flowchart showing how a program makes a request to the ABIOS.

First, in step 70, a request block for the specific request at hand is allocated and filled in. In step 71, the "current" CDA anchor pointer is accessed. In subsequent steps 72 and 73, the anchor pointer and the RB pointer are saved in the stack frame using PUSH instructions. The appropriate logical ID from the RB is accessed in step 74. (There is of course a one-to-one correspondence between logical IDs and device entries in the CDA.) In step 75, the logical ID is multiplied by 2^3 to obtain the offset of the DB pointer (four bytes/pointer, two pointers). The respective FTT pointer four bytes beyond the address of the DB pointer is accessed in step 76, and this FTT pointer is saved to the stack frame using a PUSH instruction in step 77. In step 78, the DB pointer is saved to the stack frame, also using a PUSH instruction. Finally, in step 79, the desired function is called in the FTT.

Further applications are also contemplated within the scop of the invention. For example, "patching" of the BIOS code to effect changes thereto can be achieved using the FTT pointers. That is, FTT pointers can be inserted in the CDAs which cause redirection to patch routines.

Claims

1. A computer system comprising a central processing unit (CPU) and a memory, said CPU having a first real addressing mode and a second protected addressing mode, said memory addressing modes being mutually incompatible, and characterised by a basic input/output system (ABIOS, Fig. 5) adapted to set up first and second tables in said memory containing functionally equivalent pointers compatible respectively with said first and second modes and for producing the same physical address to predetermined regions in the memory and devices attached to the system, said basic input output system being further adapted to access the first table under the first CPU mode and to access the second table under the second CPU mode.

2. A computer system according to Claim 1, in which said CPU is capable of addressing more of said memory in said second mode than in said first mode.

3. A computer system according to Claim 1 or Claim 2, in which each pointer in said first and second table comprises a lower order portion and a higher order portion, lower order portions of corresponding pointers in the first and second tables being equal to one another.

4. A computer system according to Claim 3, in which the higher order portions of the pointers in the first table are indicative of boundary addresses between fixed segments in the memory.

5. A computer system according to Claim 3 or Claim 4, in which the higher order portions of the pointers in the second table are indicative of addresses in a segment descriptor table, each segment descriptor table address containing a boundary address between fixed segments in the memory.

6. A computer system according to any of the previous claims in which said first and second tables are set up in the memory under control of a power-on self-test routine entered when the computer is switched on or reset.

7. A method of operating a computer system having a memory and a CPU capable of operating in a real and in a protected addressing mode, said real and said protected addressing modes being two mutually incompatible memory addressing modes, said method characterised by the steps of:

allocating space in said memory for first and second common data areas corresponding respectively to the first and second of said modes;

loading said first common data area with a table containing first pointers to predetermined areas of the memory and to devices attached to the system, said first positions comprising first mode addresses; and

loading said second common data area with a table containing pointers functionally equivalent to those of the first table for producing the same physical address, but comprising second mode addresses;

whereby the CPU subsequently refers to said first table for first mode addressing and said second table for second mode addressing.

8. A method according to claim 7, wherein said first mode is real mode addressing and said second mode is protected mode addressing.

9. A method according to claim 7 or claim 8, in which said pointers comprise high and low order portions, in which in the first table the high order portions relate to fixed memory segment boundaries and in the second table the high order portions relate to addresses of a description table containing fixed memory segment boundaries.

10. A method according to claim 9, wherein the low orders portions of the pointers in the second table are copied from the low order portions of the corresponding pointers in the first table.

11. A method according to any of claims 7 to 10, in which said pointers include device block pointers, function transfer table pointers, data pointers and function pointers.

Patentansprüche

1. Computersystem mit einer Zentraleinheit (CPU) und einem Speicher, bei dem die CPU einen ersten Real Adressierungsmodus und einen zweiten Protected Adressierungsmodus aufweist, die gegenseitig inkompatibel sind, gekennzeichnet durch ein BIOS (ABIOS, Fig. 5) zum Aufbau erster und zweiter Tabellen in besagtem Speicher mit funktionsmäßig gleichen Zeigern, die mit den ersten bzw. zweiten Modi kompatibel sind und zum Erzeugen derselben physikalischen Adresse zu vorbestimmten Bereichen im Speicher und den mit dem System verbundenen Einheiten, wobei das besagte BIOS außerdem so aufgebaut ist, daß es auf die erste Tabelle im ersten CPU Modus und die zweite Tabelle im zweiten CPU Modus zugreifen kann.

2. Computersystem nach Anspruch 1, in dem die besagte CPU im zweiten Modus weitere Speicherbereiche als im ersten Modus adressieren kann.

3. Computersystem nach Anspruch 1 oder 2, in dem jeder Zeiger in der ersten und zweiten Tabelle einen niedrig- und einen hochstelligen Teil aufweist, wobei die niedrigstelligen Teile entsprechender Zeiger in den ersten und zweiten Tabellen gleich sind.

4. Computersystem nach Anspruch 3, in dem die hochstelligen Teile der Zeiger in der ersten Tabelle Grenzadressen zwischen festen Segmenten im Speicher anzeigen.

5. Computersystem nach Anspruch 3 oder Anspruch 4, in dem die hochstelligen Teile der Zeiger in der zweiten Tabelle Adressen in einer Segmentbeschreibertabelle anzeigen, wobei jede Segmentbeschreibertabellenadresse eine Grenzadresse zwischen festen Segmenten im Speicher enthält.

6. Computersystem nach einem der vorhergehenden Ansprüche, in dem besagte erste und zweite Tabellen im Speicher unter Steuerung einer Power-on Self Test-Routine erstellt werden, die dann durchgeführt wird, wenn der Computer eingeschaltet oder zurückgesetzt wird.

7. Verfahren zum Betrieb eines Computersystems mit einem Speicher und einer CPU, die in einem Real und einem Protected Adressierungsmodus arbeiten kann, wobei die besagten Real und Protected Adressierungsmodi gegenseitig inkompatibel sind, gekennzeichnet durch folgende Schritte:
Zuteilung von Platz im besagten Speicher für erste und zweite gemeinsame Datenbereiche, die den ersten bzw. zweiten Modi entsprechen;
Laden des ersten gemeinsamen Datenbereichs mit einer Tabelle, die erste Zeiger auf vorbestimmte Bereiche des Speichers und auf mit

dem System verbundene Einheiten aufweist, wobei die ersten Positionen erste Modus-Adressen umfassen; sowie
Laden des zweiten gemeinsamen Datenbereichs mit einer Tabelle, die Zeiger enthält, die funktionsmäßig denen der ersten Tabelle zur Erzeugung derselben physikalischen Adresse entsprechen, aber zweite Modus-Adressen umfassen;
wobei die CPU anschließend zur Adressierung im ersten Modus auf die erste Tabelle und zur Adressierung im zweiten Modus auf die zweite Tabelle Bezug nimmt.

8. Verfahren nach Anspruch 7, wobei es sich bei dem ersten Modus um einen Real Adressierungsmodus und dem zweiten Modus um einen Protected Adressierungsmodus handelt.

9. Verfahren nach Anspruch 7 oder 8, wobei besagte Zeiger hoch- und niedrigstellige Teile aufweisen, in denen sich in der ersten Tabelle die hochstelligen Teile auf feste Speichersegmentgrenzen und in der zweiten Tabelle die hochstelligen Teile auf Adressen einer Beschreibertabelle mit festen Speichersegmentgrenzen beziehen.

10. Verfahren nach Anspruch 9, in dem die niedrigstelligen Teile der Zeiger in der zweiten Tabelle aus den niedrigstelligen Teilen der entsprechenden Zeiger in der ersten Tabelle kopiert werden.

11. Verfahren nach einem der Ansprüche 7 bis 10, in dem besagte Zeiger Einheitenblockzeiger, Funktionsübertragungstabellenzeiger, Datenzeiger und Funktionszeiger umfassen.

Revendications

1. Système d'ordinateur comprenant une unité centrale de traitement (CPU) et une mémoire, ledit CPU ayant un premier mode d'adressage réel et un deuxième mode d'adressage protégé, lesdits modes d'adressage de mémoire étant mutuellement incompatibles, caractérisé par un système d'entrée/sortie de base (ABIOS, Fig. 5) prévu pour établir une première et une deuxième tables dans ladite mémoire, contenant des pointeurs fonctionnellement équivalents compatibles respectivement avec lesdits premier et deuxième modes et produisant la même adresse physique à des régions prédéterminées de la mémoire et des dispositifs connectés au système, le dit systèm d'entrée/sortie de base étant en outre prévu pour accéder à la première table dans le premier mode de CPU et pour accéder à la deuxième table dans le deuxième mode de CPU.

2. Système d'ordinateur suivant la revendication 1, dans lequel ledit CPU est capable d'adresser une plus grande partie de ladite mémoire dans le dit deuxième mode que dans ledit premier mode.

3. Système d'ordinateur suivant la revendication 1 ou la revendication 2, dans lequel chaque pointeur desdites première et deuxième tables comprend une partie de poids faible et une partie de poids fort, les parties de poids faible des pointeurs correspondants dans les première et deuxième tables étant mutuellement égales.

4. système d'ordinateur suivant la revendication 3, dans lequel les parties de poids fort des pointeurs de la première table sont indicatives d'adresses de limites entre des segments fixes de la mémoire.

5. Système d'ordinateur suivant la revendication 3 ou la revendication 4, dans lequel les parties de poids fort des pointeurs de la deuxième table sont indicatives d'adresses dans une table de descripteurs de segment, chaque adresse de table de descripteurs de segment contenant une adresse de limite entre des segments fixes de la mémoire.

6. Système d'ordinateur suivant l'une quelconque des revendications précédentes, dans lequel les dites première et deuxième tables sont établies dans la mémoire sous la commande d'un sous-programme de test automatique à la mise sous tension, qui est lancé lorsque l'ordinateur est mis sous tension ou réinitialisé.

7. Procédé d'exploitation d'un système d'ordinateur ayant une mémoire et une unité centrale CPU capable de fonctionner en un mode d'adressage réel et en un mode d'adressage protégé, lesdits modes d'adressage réel et protégé étant deux modes d'adressage de mémoire mutuellement incompatibles, ledit procédé étant caractérisé par les étapes de :
affectation d'espace dans ladite mémoire pour une première et une deuxième zones de données communes correspondant respectivement aux premier et deuxième desdits modes ;
chargement de ladite première zone de données communes avec une table contenant des premiers pointeurs à des zones prédéter-

minées de la mémoire et à des dispositifs raccordés au système, lesdits premiers pointeurs comprenant des adresses de premier mode ; et

chargement de ladite deuxième zone de données communes avec une table contenant des pointeurs fonctionnellement équivalents à ceux de la première table pour produire la même adresse physique, mais comprenant des adresses de deuxième mode ;

de sorte que le CPU consulte ensuite ladite première table pour l'adressage dans le premier mode et ladite deuxième table pour l'adressage dans le deuxième mode.

8. Procédé suivant la revendication 7, dans lequel ledit premier mode est un adressage en mode réel et ledit deuxième mode est un adressage en mode protégé.

9. Procédé suivant la revendication 7 ou la revendication 8, dans lequel lesdits pointeurs comprennent des parties de poids fort et des parties de poids faible, les parties de poids fort de la première table désignant des limites de segments fixes de la mémoire et les parties de poids fort de la deuxième table désignant des adresses d'une table de description contenant des limites de segments fixes de la mémoire.

10. Procédé suivant la revendication 9, dans lequel les parties de poids faible des pointeurs de la deuxième table sont copiées à partir des parties de poids faible des pointeurs correspondants de la première table.

11. Procédé suivant l'une quelconque des revendications 7 à 10, dans lequel lesdits pointeurs comprennent des pointeurs de bloc de dispositif, des pointeurs de table de transfert de fonction, des pointeurs de données et des pointeurs de fonction.

FIG.1

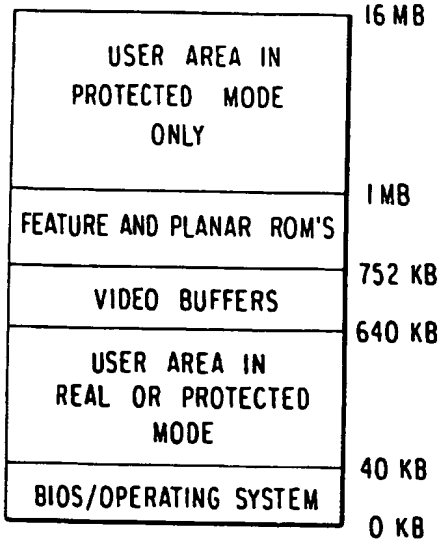


FIG.5

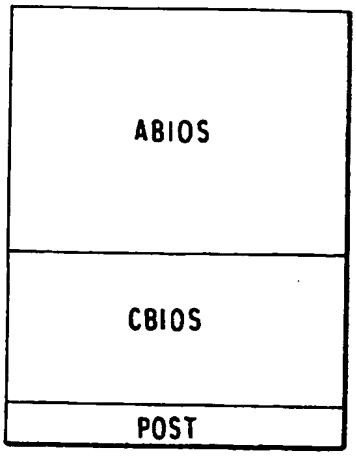


FIG.2

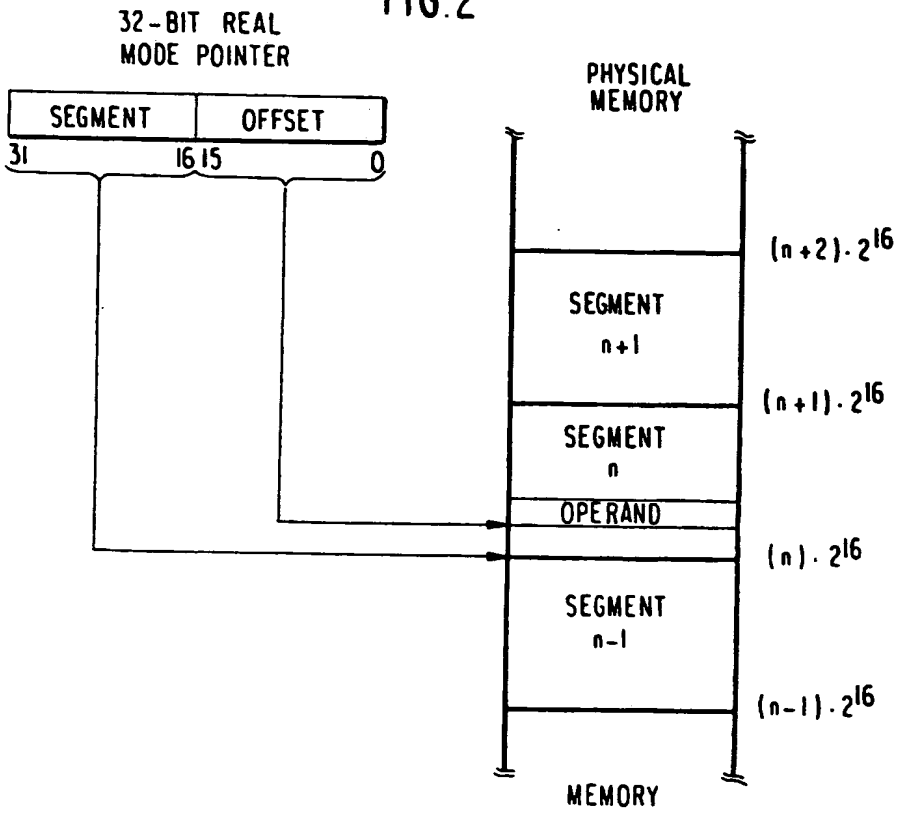


FIG.3

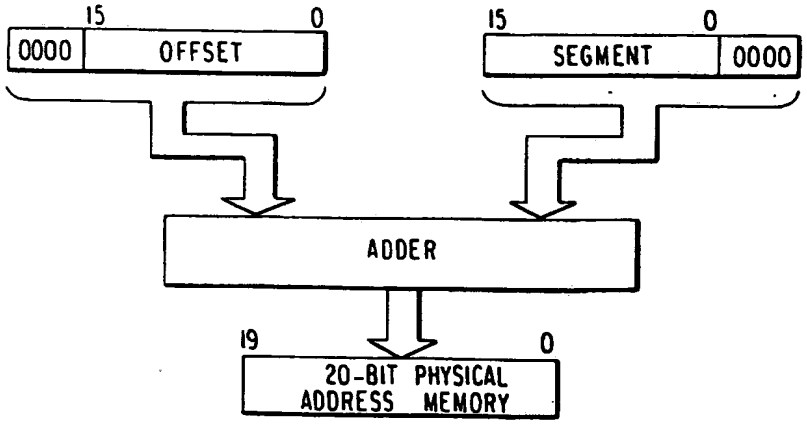

15
0
0000
OFFSET
15
0
SEGMENT
0000
ADDER
19
0
20-BIT PHYSICAL ADDRESS MEMORY

FIG.4

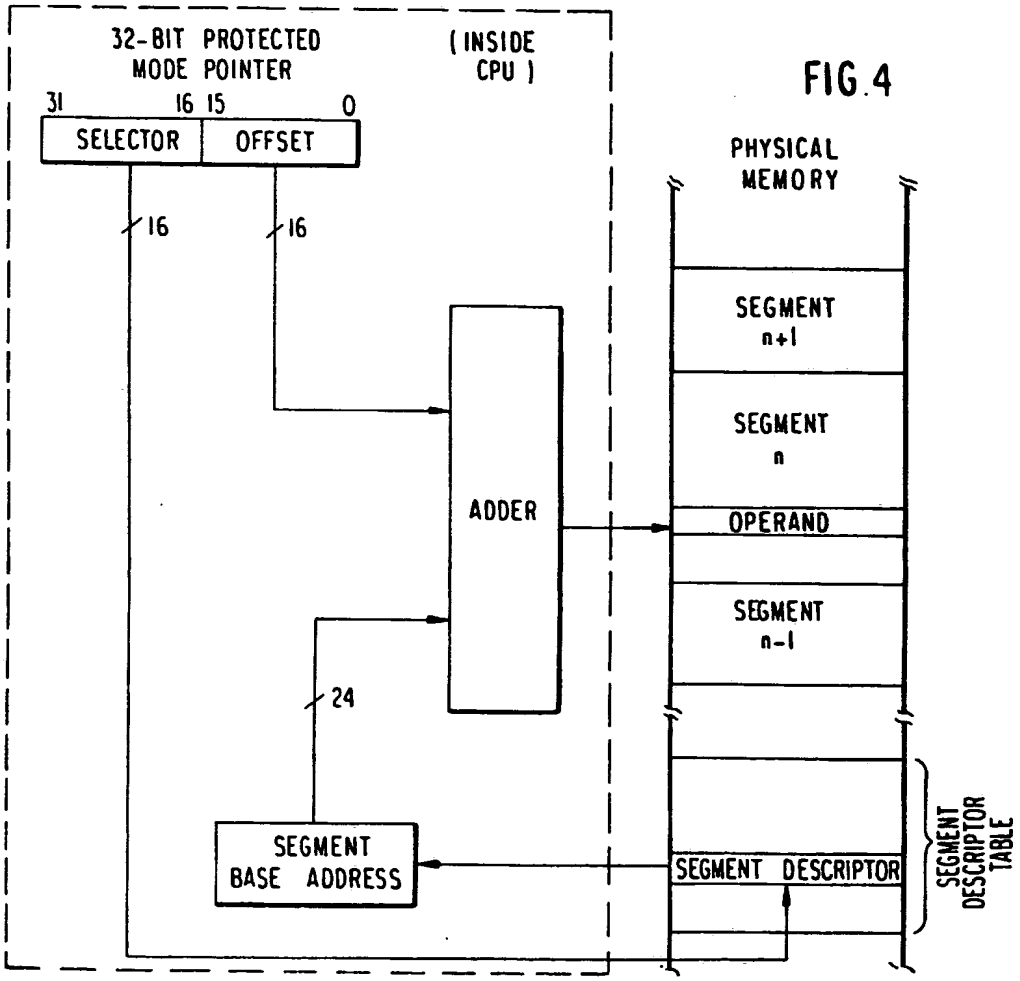
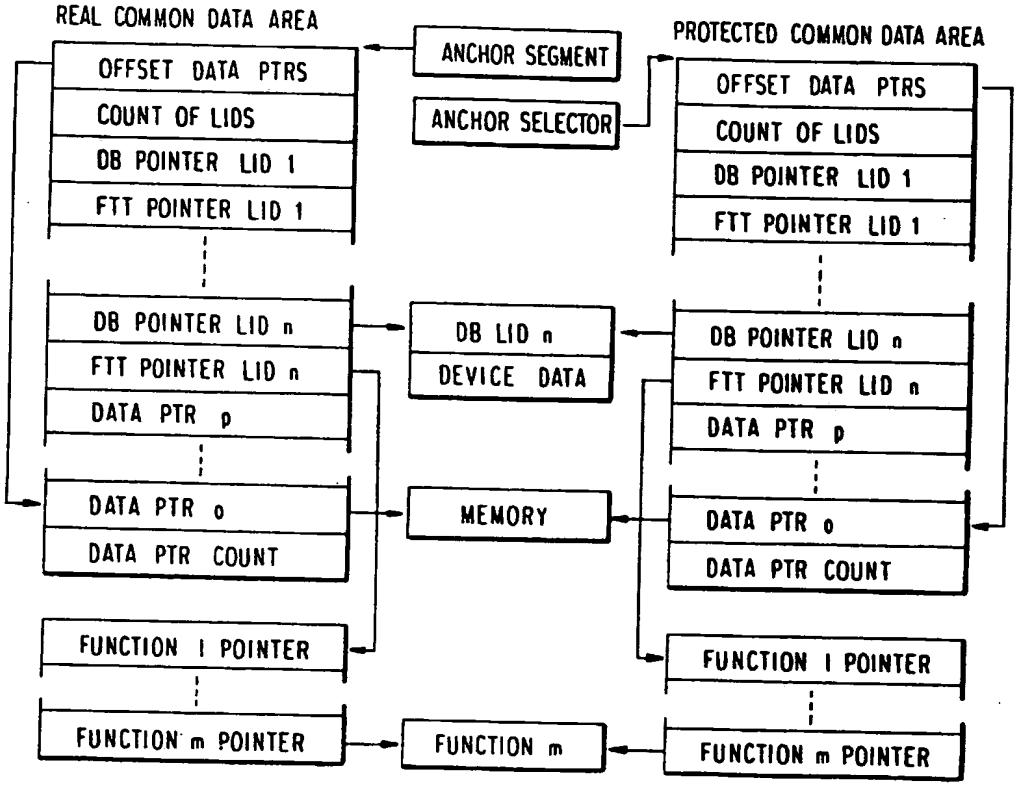

32-BIT PROTECTED MODE POINTER
(INSIDE CPU)
31
16 15
0
SELECTOR
OFFSET
16
16
ADDER
24
SEGMENT BASE ADDRESS
PHYSICAL MEMORY
SEGMENT n+1
SEGMENT n
OPERAND
SEGMENT n-1
SEGMENT DESCRIPTOR
SEGMENT DESCRIPTOR TABLE

FIG.6


REAL COMMON DATA AREA
OFFSET DATA PTRS
COUNT OF LIDS
DB POINTER LID 1
FTT POINTER LID 1
DB POINTER LID n
FTT POINTER LID n
DATA PTR p
DATA PTR o
DATA PTR COUNT
FUNCTION 1 POINTER
FUNCTION m POINTER
ANCHOR SEGMENT
ANCHOR SELECTOR
DB LID n
DEVICE DATA
MEMORY
FUNCTION m
PROTECTED COMMON DATA AREA
OFFSET DATA PTRS
COUNT OF LIDS
DB POINTER LID 1
FTT POINTER LID 1
DB POINTER LID n
FTT POINTER LID n
DATA PTR p
DATA PTR o
DATA PTR COUNT
FUNCTION 1 POINTER
FUNCTION m POINTER

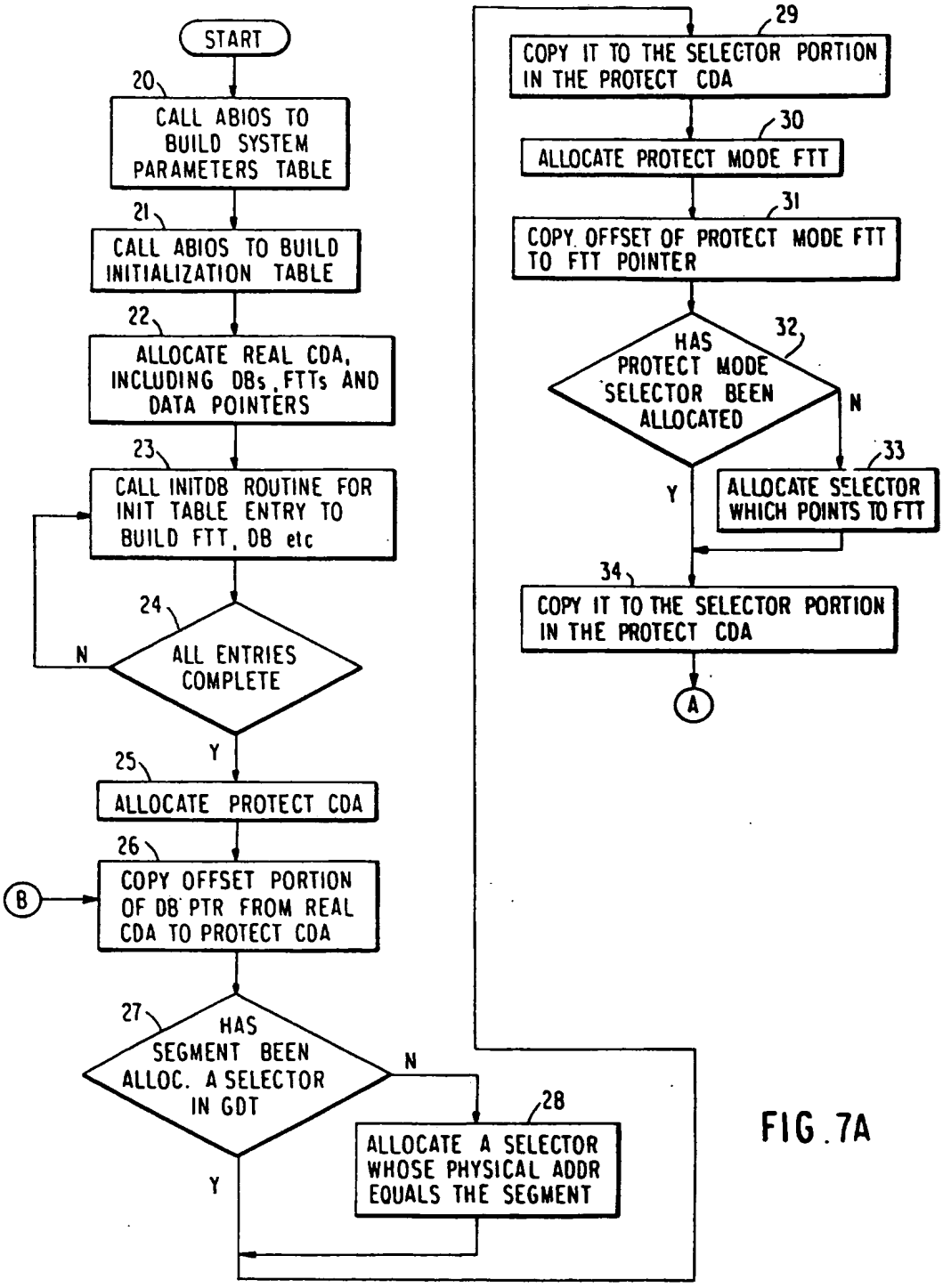

START
20
CALL ABIOS TO BUILD SYSTEM PARAMETERS TABLE
21
CALL ABIOS TO BUILD INITIALIZATION TABLE
22
ALLOCATE REAL CDA, INCLUDING DBs, FTTs AND DATA POINTERS
23
CALL INITDB ROUTINE FOR INIT TABLE ENTRY TO BUILD FTT, DB etc
24
ALL ENTRIES COMPLETE
N
Y
25
ALLOCATE PROTECT CDA
26
B
COPY OFFSET PORTION OF DB PTR FROM REAL CDA TO PROTECT CDA
27
HAS SEGMENT BEEN ALLOC. A SELECTOR IN GDT
N
Y
28
ALLOCATE A SELECTOR WHOSE PHYSICAL ADDR EQUALS THE SEGMENT
29
COPY IT TO THE SELECTOR PORTION IN THE PROTECT CDA
30
ALLOCATE PROTECT MODE FTT
31
COPY OFFSET OF PROTECT MODE FTT TO FTT POINTER
32
HAS PROTECT MODE SELECTOR BEEN ALLOCATED
N
Y
33
ALLOCATE SELECTOR WHICH POINTS TO FTT
34
COPY IT TO THE SELECTOR PORTION IN THE PROTECT CDA
A

FIG. 7A

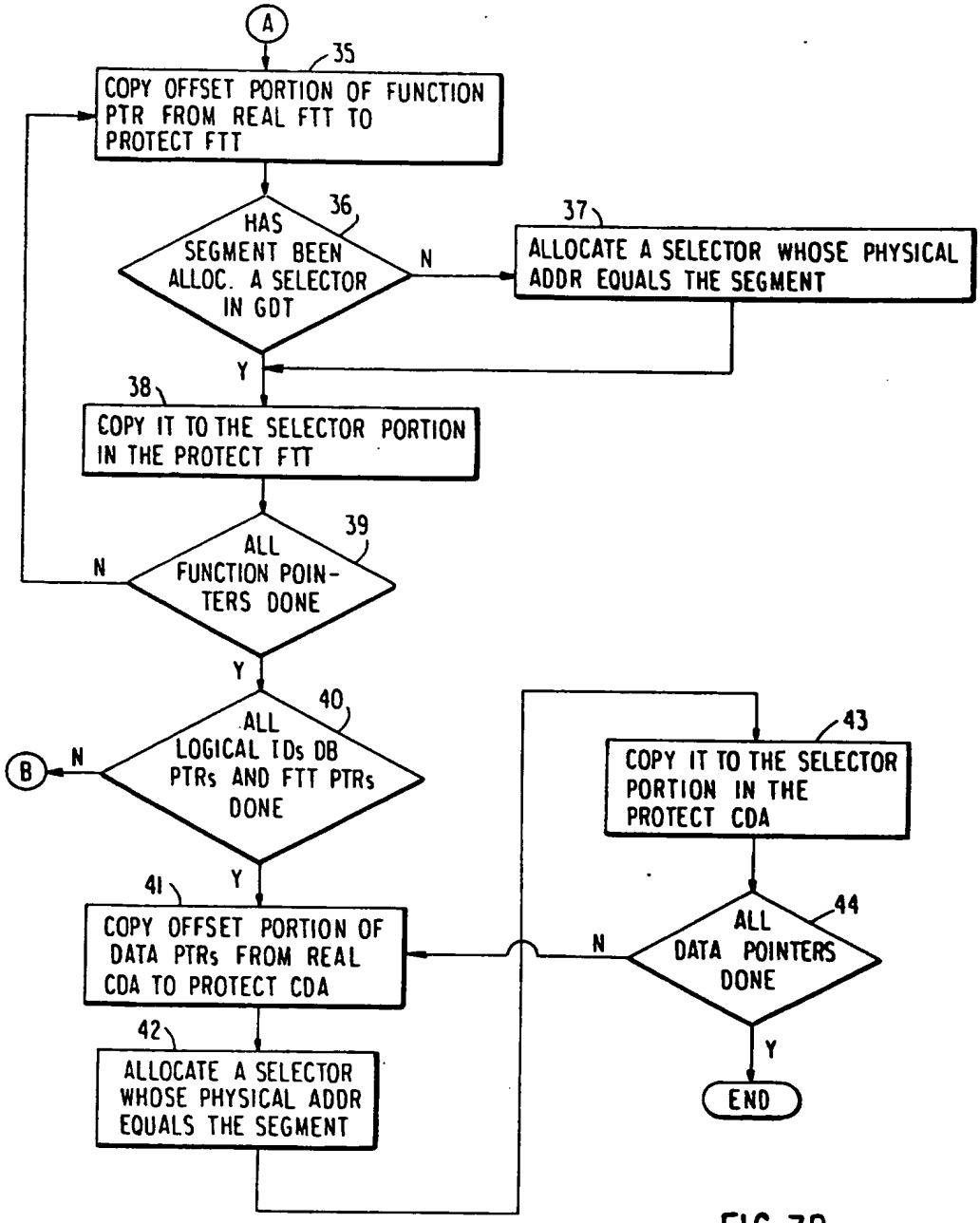

A
35
COPY OFFSET PORTION OF FUNCTION PTR FROM REAL FTT TO PROTECT FTT
36
HAS SEGMENT BEEN ALLOC. A SELECTOR IN GDT
N
37
ALLOCATE A SELECTOR WHOSE PHYSICAL ADDR EQUALS THE SEGMENT
Y
38
COPY IT TO THE SELECTOR PORTION IN THE PROTECT FTT
39
ALL FUNCTION POINTERS DONE
N
Y
40
ALL LOGICAL IDs DB PTRs AND FTT PTRs DONE
B
N
Y
41
COPY OFFSET PORTION OF DATA PTRs FROM REAL CDA TO PROTECT CDA
42
ALLOCATE A SELECTOR WHOSE PHYSICAL ADDR EQUALS THE SEGMENT
43
COPY IT TO THE SELECTOR PORTION IN THE PROTECT CDA
44
ALL DATA POINTERS DONE
N
Y
END

FIG.7B

FIG.8

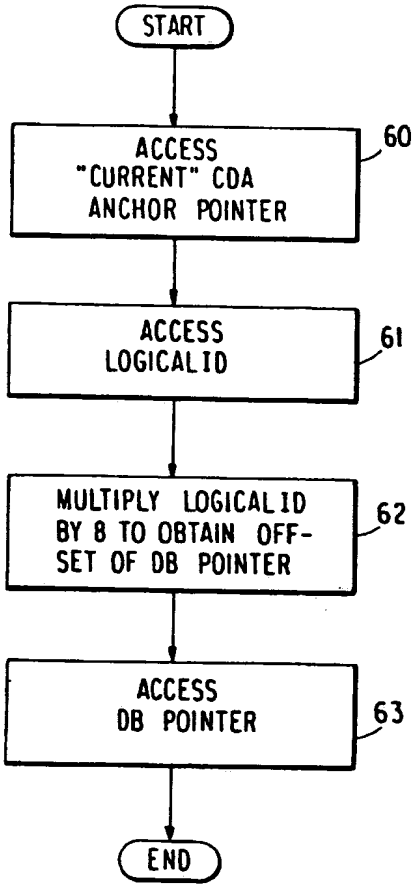
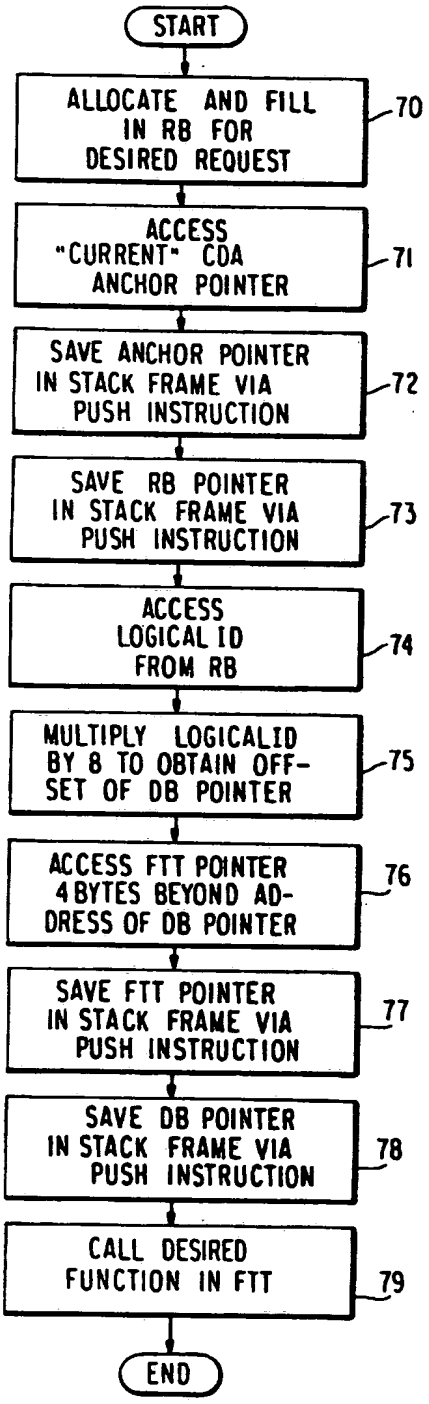

START
ACCESS "CURRENT" CDA ANCHOR POINTER
60
ACCESS LOGICAL ID
61
MULTIPLY LOGICAL ID BY 8 TO OBTAIN OFF-SET OF DB POINTER
62
ACCESS DB POINTER
63
END

FIG.9


START
ALLOCATE AND FILL IN RB FOR DESIRED REQUEST
70
ACCESS "CURRENT" CDA ANCHOR POINTER
71
SAVE ANCHOR POINTER IN STACK FRAME VIA PUSH INSTRUCTION
72
SAVE RB POINTER IN STACK FRAME VIA PUSH INSTRUCTION
73
ACCESS LOGICAL ID FROM RB
74
MULTIPLY LOGICALID BY 8 TO OBTAIN OFF-SET OF DB POINTER
75
ACCESS FTT POINTER 4 BYTES BEYOND ADDRESS OF DB POINTER
76
SAVE FTT POINTER IN STACK FRAME VIA PUSH INSTRUCTION
77
SAVE DB POINTER IN STACK FRAME VIA PUSH INSTRUCTION
78
CALL DESIRED FUNCTION IN FTT
79
END